UNIVERSITY OF GLASGOW

COMPUTING DEPARTMENT

MULTUM OPERATING SYSTEM MEMO NO. 3 (17/8/72)

OUTLINE SPECIFICATION OF A VIRTUAL I/O SYSTEM

1. Introduction.

1.1 General.

The purpose of this memo is to explain the role of the Virtual I/O Control System (VIOC) in the community of processes which constitute the operating system. The nature of the VIOC services is outlined, but little consideration is given to questions of implementation because there is not sufficient information available yet on the facilities of the ALP2/3 executive.

1.2 The I/O hierarchy.

The I/O system is a "layered" hierarchy, each "layer" consisting of routines which implement a level of abstraction for the layer above. The lowest "layer" contains the routines which interact directly with the hardware, and is called the physical I/O controller (PIOC). PIOC services peripheral interrupts, vets and initiates transfers, creates IOP programs, handles data and command-chaining and "knows" absolute device addresses.

Built directly on to PIOC, the logical I/O controller (LIOC) achieves independence from physical device addresses for the layers above it. Each user of LIOC has a (probably empty) set of devices of each type that LIOC supports in the installation. Requests to LIOC quote only the device type and the number of the device in the user's set of that type. Such (type, number) pairs are known as "logical units". The correspondence between a logical unit and a physical unit is determined by LIOC and depends on which peripherals are free when the correspondence is established, specific requirements for the logical unit (e.g. "logical lineprinter must have at least 120 columns per line"), and operator action (e.g. a logical unit is moved from one tape reader to another as a result of a breakdown). It appears that ICS have combined the PIOC and LIOC layers when implementing their "device drivers".

If the I/O hierarchy ended at this level user programs would be inflexibly bound to specific choices of peripheral device types. This is undesirable even in simple batch installations, rendering their programs unusable in all circumstances where no device of the required type is available (e.g. because of hardware breakdown or reservation by other programs). Transfer of programs between installations is also inhibited. In multiaccess installations this restriction is intolerable, as it becomes impossible conveniently to write a program and debug it interactively (using terminal and file I/O) then run it for production in the background (using card and printer I/O instead).

Relief comes from adding a further layer to the I/O hierarchy. The virtual I/O controller (VIOC) implements a level of abstraction above which the user is freed from device characteristics. Beneath VIOC the operations are device-dependent, above it all transfers manipulate byte streams or sector streams. In fact, we go even beyond this and allow VIOC to communicate either with peripheral datasets (via LIOC) or with backing-store datasets (via the filing system). As a convenience to the user, and to make for maximal compatibility, VIOC recognises and caters for a variety of record formats in serial processing. More complicated access methods and/or formatted direct-accessing must be provided for by an access method package in a new layer of the I/O hierarchy between VIOC and the user.

2. VIOC.

2.1 Definitions.

(a) A virtual unit (VU) is a file or a document or a dummy. All operations are equally possible for a dummy VU - all are ignored!

(b) A file is a virtual backing-store and can support serial-access or direct-access transfers (serial-only in the case of magnetic tape files).

(c) A document is a virtual peripheral and can support only serial-access transfers. Magnetic tapes are considered to be documents if not included in the filestore.

(d) A dataset is the object accessed (via a VU) as a file or as a document.

(e) A dataset has a standard format determined by its physical representation. The format of a peripheral dataset is fixed solely by the hardware. The format of a backing-store dataset is fixed by the hardware and/or software conventions.

Three standard formats are defined - U, F(k) and V(k) - the latter pair admitting two variations.

(i) A dataset has U format if its records do not conform to either of the alternatives (U for "undefined").

(ii) A dataset has F(k) format if its records are structured according to the VIOC standard for fixed-length records of exactly k bytes.

(iii) A dataset has V(k) format if its records are structured according to the VIOC standard for variable-length records of at most k bytes.

For the variations, see 2.4 below.)

2.2 Virtual unit controllers.

A virtual unit controller (VUC) is an activity set up by VIOC to vet and execute requests from a user process for operations on one of its virtual units. There are two classes of VUC - those which can carry out a serial transfer only of an integral number of bytes and those which can carry out either serial or direct-access transfers of an integral number of sectors. We call the former byte VUCs (BVUCs) and the latter sector VUCs (SVUCs). Thus a BVUC may be responsible for a document or a file, whereas a SVUC may be responsible only for a file.

A VUC is established by the system on behalf of a user process which must specify the actual dataset involved and the mode of attachment required. The dataset specification consists of a filename, a document name, or a logical unit description. The mode of attachment indicates the access rights required (e.g. read/write/append), whether serial or direct access is required, whether the transfer unit is a sector or a byte, and (in the serial-byte case) whether 8-bit binary bytes or 7-bit character bytes are involved. An additional option for character transfers indicates whether file-substitution is to operate.

The response to a request to establish a VUC indicates the outcome. Among the possibilities are "OK", "actual dataset reserved", "actual dataset does not exist", "actual dataset cannot support requested modes" and "garbled request rejected".

2.3 Logical unit descriptions.

The use of a logical unit description (LUD) allows the user to control the selection of an actual unit by demanding that it possesses certain attributes or properties. Each property is encoded in a suitable fashion and the LUD consists of a list of the properties required. By this means a user process would have the power to indicate that its logical printer must have at least 120 print positions and the full 96-graphic character set, or that if there was a choice of slow and fast tape readers then the slow device would suffice. The usefulness of LUDs is further enhanced by having the VIOC indicate, when a VUC is established, what the LUD of the actual unit is.

2.4 Spanning.

Datasets of F(k) and V(k) formats can be further classified according to whether they employ "record spanning". A dataset is said to have a spanned format (denoted FS(k) and VS(k)) if records which are too large to fit into one transfer unit (a sector on discs, a block on magnetic tapes) are split across adjacent transfer units (thus "spanning" the gap). Spanning is transparent to users of VIOC, but of course it is possible to specify it on output. It is expected that spanning will be normal for disc datasets (because of the small transfer unit) and exceptional for magnetic tapes (due to the ability to have blocks of varying length).

2.5 VU operations.

The transfer operations and the skipping and positioning operations are fundamental to the use of a VU.

The length of a transfer or skip is given in terms of a number of bytes, records or sectors according as we have a BVUC or a SVUC. Further alternatives depend on whether serial or direct access is employed, and on whether inter-record breaks in the data are signalled to/by the user process.

2.5.1 BVUC operations.

Only serial transfers are possible, but each read/write can operate in one of two alternative manners:

(a) The operation commences on a new record, discarding any unread input, or, packing F(k) output records with NULL characters (all zero)/truncating V(k) output records.

(b) The operation continues with any unread or unwritten portion of the current record, taking a new record only when necessary.

We refer to option (a) as record mode and to option (b) as stream mode.

Forward and backward skipping operations also have record and stream options. A skip in record mode traverses the given number of records, while a skip in stream mode traverses the given number of bytes. Positioning operations are generally invalid for BVUCs, the only exception being reset (rewind) which returns the access mechanism to the beginning of the dataset and is operative only on files and magnetic tape documents.

2.5.2 SVUC operations.

The user-oriented byte-stream and record operations are not available with SVUCs, which work quite independently of format considerations. However, we have to consider serial and direct-access cases.

2.5.2.1 Serial SVUCs.

The only operations available with serial SVUCs are "read/write/skip forward over the next n sectors", "skip backward over the last n sectors" and reset (rewind), where n is a parameter of the operation.

2.5.2.2 Direct-access SVUCs.

All the serial SVUC operations are permitted, and we have in addition "set to sector m" which is an abstraction of the hardware "seek" For convenience "set" can be combined with read/write/skip forward/skip backward, giving a set of operations of the form "set to sector m and operate on n sectors".

2.5.3 Serial access to U format datasets.

When the actual dataset has U format the record mode and stream mode operations behave identically (the dataset is treated as one huge record), except that record mode skipping is prohibited.

2.6 File substitution.

An invaluable feature of the Egdon system is "file substitution", a facility whereby the input routines can detect a certain control card (a "*SUBSTITUTE" card) bearing a file name. Further input is then taken from the named file until the end of file condition is raised, at which point the routines revert to the original source of data. It is possible to have many *SUBSTITUTE cards in an input deck, and it is possible for a substituted file itself to contain *SUBSTITUTE cards (to a maximum permitted depth of nesting).

VIOC will support an extension of this idea, which will enable certain strings to be passed as parameters of the substituted file and inserted therein in lieu of appropriate formal parameter markers. (In the absence of any actual parameter the formal parameter marker is deleted. It is possible to suppress the parameter-replacement mechanism and to change the formal parameter warning character.) File substitution can be specified to VIOC only for BVUCs operating in character input mode.

2.7 Spooling.

In a previous document (the O.S. Outline Spec.) it was stated that the appropriate place to incorporate "spooling" was in LIOC. On reflection, it is now clear that spooling is best handled by VIOC. This enables the spooling mechanisms to use the filing system and LIOC services without doing violence to the hierarchical structure of the I/O system - a feature which should considerably facilitate the incorporation and amendment of spooling routines.

However, it should be pointed out that spooling as originally conceived is not entirely appropriate in the presence of a disc filing system. Some refinement of our ideas on spooling is necessary before firmer recommendations can be made, but we can note here that the combination of file substitution (see above) with a "job loader" that pre-reads batch jobs into a job-definition file may make the concept obsolete!